PDP-1 COMPUTER
ELECTRICAL ENGINEERING DEPARTMENT
M.I.T.
CAMBRIDGE 39, MASSACHUSETTS

PDP-11-1

PROGRAMMING THE PDP-1 COMPUTER

J. B. Dennis

July 29, 1963

Introduction

The PDP-1 is a high-speed, general purpose, stored program calculator. The fundamental unit of data is an 18-digit binary word. The PDP-1 may be used to mechanize any computation or processing procedure which can be expressed in terms of a sequence of arithmetic and logical operations on these quantities. The circuitry of any machine which is to be used in this manner must contain four basic elements. First, there must be components which can perform the arithmetic and logical operations required for a computation. This collection of hardware is commonly known as the arithmetic element of a computer. Secondly, there must be components which will hold data words until needed in the computation and save partial results for later reference. This is the memory element. The most important property of contemporary high-speed calculators is that the algorithm, or specification of the sequence of logical and arithmetic steps required to perform a computation, is also stored in the memory element. The collection of memory words which specifies an algorithm is known as a program in machine language. The third important section of a computer is a group of devices which provide for communication between the machine and the user of other entities of the outside world. This is the input-output element of a computer and typically contains means of accepting information from punched cards or paper tape, and displaying results via alphanumeric printing or graphically with a cathode ray tube or plotting board.

Finally, a computer must contain components which can examine the algorithm specification in the memory element, and translate this specification into the appropriate sequence of actions in the arithmetic element, memory element and input-output element. The relationships among the four elements are indicated in Figure 1.

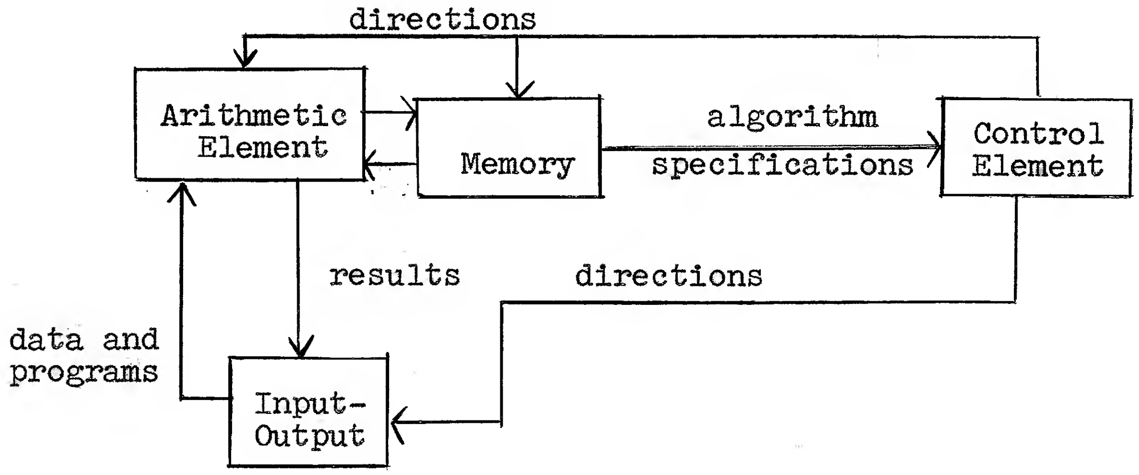


Figure 1--The Four Basic Elements of a Stored Program Computer

The system organization of the PDP-1 is shown in Figure 2. The accumulator (AC), memory buffer register (MB) and the in-out (IO) are flip-flop registers which can hold 18-digit binary words. These registers constitute the arithmetic element of the machine. All arithmetic and logical operations performed by the computer are executed on the contents of these registers.

The input-output element of the PDP-1 includes a paper tape reader which is used to communicate programs and data to the machine. The typewriter keyboard may be used to direct a program to select a particular computation when many possibilities are present, and can also be used to change parameters or to modify a program. Calculated results may be presented to the user in a numerical form by printing with the typewriter, or in graphical form on the oscilloscope display. Results may also be punched into paper tape for further processing at a later time. A unique

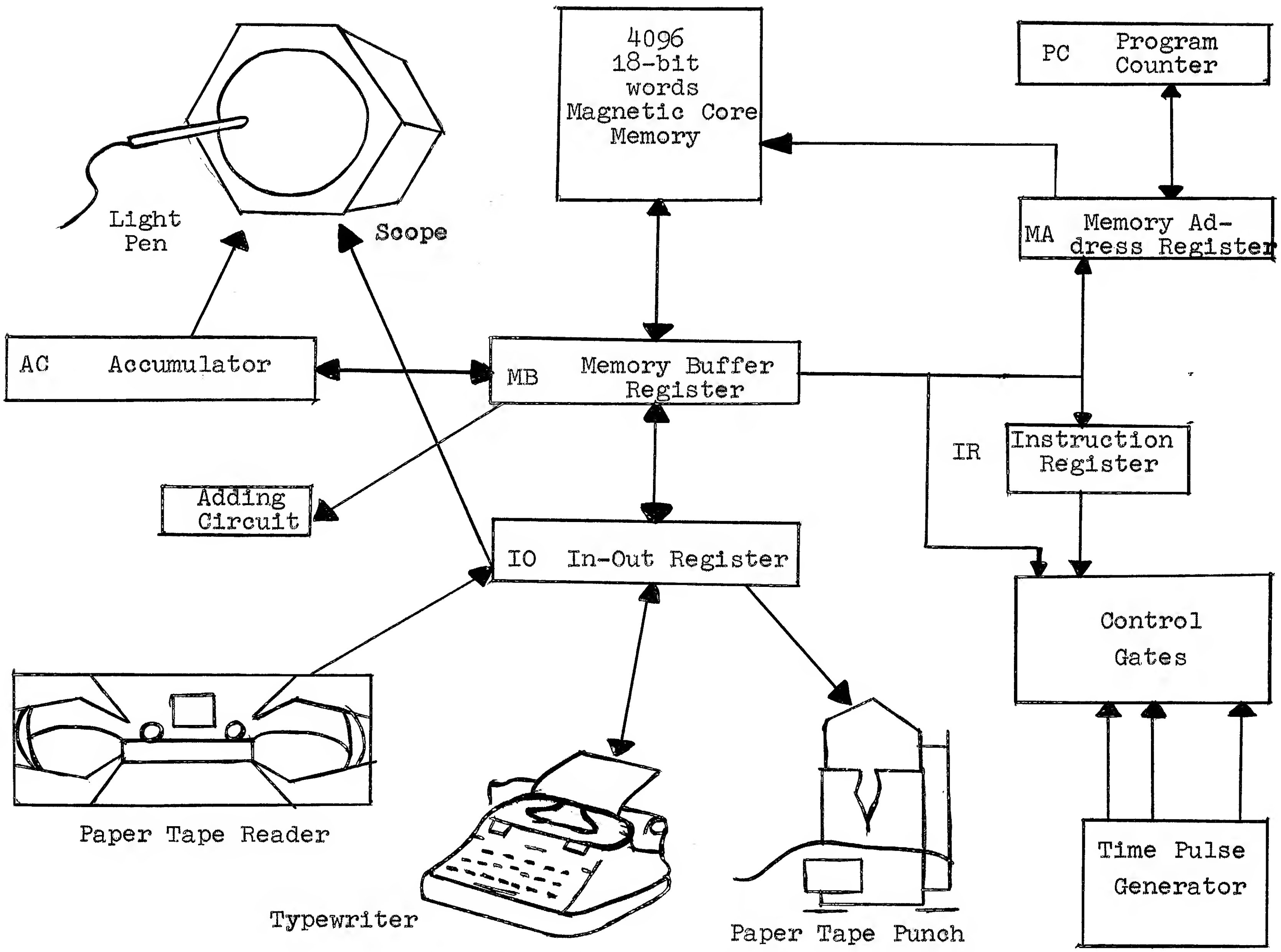


Figure 2--PDP System Organization

input device available with the PDP-1 is the light pen which is used in conjunction with the oscilloscope to read in empirical curves or to direct a program to take a specified action. Means are also provided to transfer information to and from users' equipment.

The PDP-1 is equipped with a ferrite core memory having 4096 registers, each capable of holding an 18 binary digit word. With each register a unique address is associated which, in octal notation* may range from 0 to 7777. One memory cycle time, or five microseconds, is required whenever a word is read from a memory register into the memory buffer register, or a word in the MB is placed in a memory register. The memory register involved is always specified by the address contained in the memory address register, (MA).

The control element of the PDP-1 consists of two 12-bit registers, the program counter (PC) and the memory address register (MA), and a 6-bit register called the instruction register (IR). The control element also contains a time pulse generator which produces a sequence of 10 pulses which is repeated for each five microsecond memory cycle. Gate circuits in the control element allow these pulses to cause various actions in the machine according to the contents of the instruction register, and in some cases, the memory buffer register.

Actions which the control element may effect include the following:

1. transfer a word to the arithmetic element from a specified location in memory.

*All numerical quantities subsequently used in this memo are in octal notation unless otherwise indicated by the appropriate subscript.

2. transfer a word to a specified location in storage from the arithmetic element.
3. perform a logical or arithmetic operation within the arithmetic element, such as clear, add, etc.
4. transfer data from an input device to the arithmetic element, e.g., read a line of punched tape via the photo electric tape reader.
5. transfer data to an output device, e.g., type a character by means of the typewriter.
6. transfer the contents of a memory register to IR and MA for interpretation by the control element.

Action of Control

In the PDP-1 computer, the words making up the program for a particular computation are called instructions and are interpreted one at a time by the control element. The program counter (PC) indicates to the control element which memory register contains the next instruction to be interpreted and executed. The contents of the PC is indexed for each instruction performed so that instructions in successive memory locations are executed in sequence. (There are exceptions to this which will be taken up later.)

To show the steps by which an instruction is executed by the control element, consider the following example: It is desired to add the number 45 to the contents of the AC which is initially 13. To be able to accomplish this, the 18-digit binary representation of 45 must be in some memory location, say register 100. The instruction which will cause the addition must specify two things: First, it must give the address of the register whose contents are to be added. Second, it must specify that addition is to be performed rather than some other operation. This is accomplished by employing the instruction format given in Figure 3.

The operation code consists of six binary digits and could select one out of 64_{10} different possibilities. For instance, an addition instruction has the operation code 100000_2. The address section of an instruction usually specifies which memory register is involved in the operation.

To return to our example, the instruction to add the contents of register 100 to the present contents of the AC would be represented by the octal number

400100

In order to be executed as part of a program, this instruction must be stored in some memory register of the machine, say register 200. We may indicate the initial status of memory registers of interest by writing the address followed by a slash, then the contents of the register. Thus we have

```
100/    45
200/    400100
```

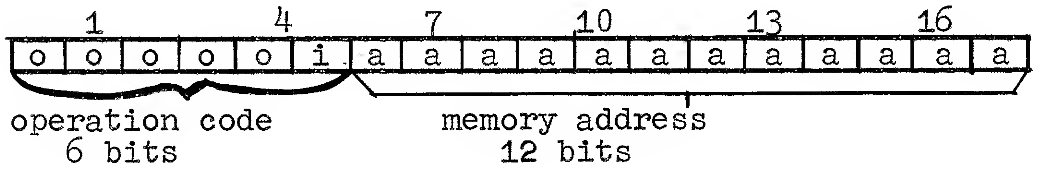


Figure 3--PDP-1 Instruction Format

It is evident now that the execution of the addition instruction will require at least two memory cycle times. One is necessary to read the instruction from register 200 and transfer it to the control element. This is known as cycle zero. A second memory cycle is then needed to read the contents of register 100 into the MB so that it may be added to the AC. The actual sequence of events is given below. In this description, the symbol AC will mean "the contents of the accumulator", and likewise for MB, PC, and MA. An arrow means "replaces". The abbreviation CM<MA> refers to the core memory register selected by MA.

<u>Initial Conditions</u>:		
	AC/	13
	PC/	200
	200/	400100
	100/	45

<u>Cycle zero</u>--fetch instruction

1. PC→MA MA/ 200

Memory register 200 containing the instruction to be executed is selected.

2. CM<MA>→MB MB/ 400100

The instruction to be executed is read from memory register 200.

3. MB_{0-5}→IR IR/ 100000_2= 40

The instruction code for <u>add</u> is placed in the control element.

4. PC + 1→PC PC/ 201

The program counter is indexed so that the instruction in register 201 will be interpreted next.

<u>Cycle one</u>--execute instruction

1. MB_{6-17}→MA MA/ 100

Memory register 100 containing the addend is selected.

2. CM<MA>→MB MB/ 45

The addend is read into the memory buffer register.

3. MB + AC→AC AC/ 60

The addition is performed

Final Conditions:		
	AC/	60
	PC/	201
	200/	400100
	100/	45

The control element will then return to the sequence of steps under cycle zero, which is essentially the same for all instructions.

The PDP-1 Instruction Code--Information Transfer Instructions

Five of the PDP-1 instruction codes provide for the transfer of information between the registers of the arithmetic element and the memory element. These instructions are listed in Table 1. In the "symbol" and "octal value" columns of the chart, the symbol a represents an arbitrary memory address ranging from 0 through 7777. It is convenient to think of the symbols for the operations codes as having a value as given in the second column. The last column of the chart gives a symbolic description of the execution of the instruction where the notation is the same as used in the previous section. For example, the description of the lac instruction should be read as "The contents of the a th memory register becomes the new contents of the accumulator." The halt instruction is included in the table so that we will have a means of stopping the computation at the completion of our examples.

Suppose as part of an algorithm, it is desired to interchange the values assigned to two quantities named x and y.

We could indicate this in an algorithm diagram as shown in figure 4a. So that we can perform this computation with the available PDP-1 instructions, we may elaborate on this as shown in Figure 4b.

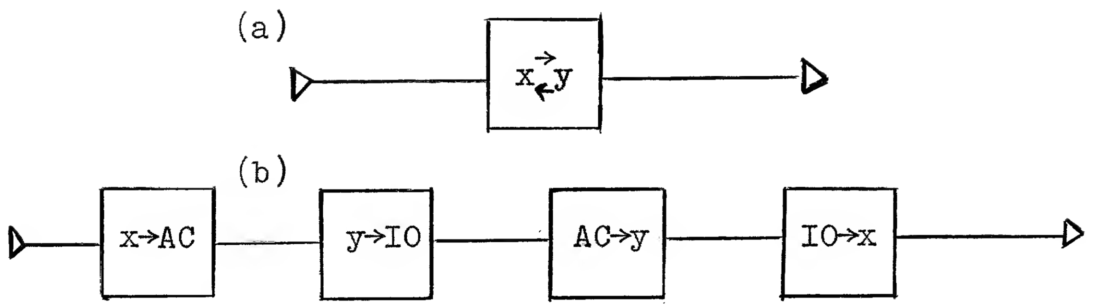


Figure 4--Logical Diagrams for Interchange

To convert this into a computer program, we must think of the names x and y as being associated with specific registers in the memory of our computer. The value assigned to a name by an algorithm is then the contents of the register associated with the name. It is convenient for programming purposes to think of name or symbol x as having a value equal to the address of the register containing the quantity named x, and similarly for y. The notation in Figure 4a can then be interpreted to mean, "Interchange the contents of registers x and y."

With this understanding, the computation specified in Figure 4b may be performed by the following symbolic program for the PDP-1 computer.

```
20/         lac x       x→AC
            lio y       y→IO
            dac y       AC→y
            dio x       IO→x
            hlt         Halt
```

The beginning programmer should be very careful to distinguish between the value assigned to a name by an algorithm (meaning the contents of a register), and the value of the name itself (meaning the address of the register).

Table 1

Symbol	Octal Value	Name	Description
lac a	200000 + a	Load Accumulator	CM<a>→AC
	Load the accumulator from register a.		
lio a	220000 + a	Load in-out	CM<a>→IO
	Load the in-out register from register a.		
dac a	240000 + a	Deposit Accululator	AC→CM<a>
	Copy the number in the accumulator into memory register a.		
dio a	320000 + a	Deposit in-out	IO→CM<a>
	Copy the number in the in-out register into memory register a.		
dzm a	340000 + a	Deposit zero in memory	0→CM<a>
	Place plus zero in memory register a.		
hlt	760400	Halt	------
	Halt execution of instructions. Execution will continue in sequence when the continue lever is pressed.		

The notation "20/" on the first line of the program indicates that the first instruction lac x is to be placed in register 20 of the PDP-1. If x and y are associated with registers 100 and 101 of the memory, the program would appear in memory as the following sequence of octal numbers:

```
20/     200100
21/     220101
22/     240101
23/     320100
24/     760400
```

These numbers may be found by adding together the value of the symbols making up the instruction, for instance

lio y = 220000 + 101 = 220101

It is helpful in this regard to think of the space between two symbols as equivalent to a plus sign.

Note that it is impossible to tell whether the contents of any given register is meant as an instruction or a word of data. The distinction is that only those words meant as instructions ever enter the PDP-1 control element to cause the corresponding action to be performed. It frequently happens, however, that an error in programming leads the computer to interpret a succession of data words as instructions. When this happens, the consequences may be startling!

Arithmetic Instructions

The PDP-1 computer performs arithmetic operations on quantities using the one's complement form for negative numbers. In the following chart the plus sign is used to indicate one's complement addition of 18-bit binary quantities and the minus sign indicates that one's complement of the following quantity is to be taken.

Symbol	Octal Value	Name	Description
add a	400000 + a	Add	AC + CM<a>→AC
	Add the contents of register <u>a</u> to the accumulator.		
sub a	420000 + a	Subtract	AC + (-CM<a>)→AC
	Subtract the contents of register <u>a</u> from the accumulator.		
cma	761000	Complement AC	-AC→AC
	Complement the contents of the accumulator.		

Note that there are two representations for zero--plus zero (000000), and minus zero (777777). The logic of the PDP-1 is arranged so that a zero result of addition or subtraction is always represented as plus zero.

To illustrate the computation shown in Figure 5

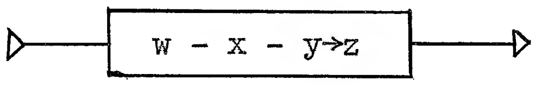


Figure 5--Simple Arithmetic Computation

Simple arithmetic computation could be accomplished by either of the following programs:

```
(a)     20/  lac w                (b)     20/  lac x
             sub x                             add y
             sub y                             cma
             dac z                             add w
             hlt                               dac z
                                               hlt
```

Decision Instructions

With our catalog of instructions so far, the control element of PDP-1 steps through memory one register at a time interpreting each word as an operation in the arithmetic element or a transfer to or from memory. There is no means of interrupting the sequence so that certain groups of instructions may be repeated, and there is no provision for testing instructions which could direct control along different paths depending on a computer result.

In the PDP-1 machine these features are provided by the jump instruction together with skip commands. These instructions are given in the chart below:

Symbol	Octal Value	Name	Description
jmp a	600000 + a	Jump	a→PC
	Take next instruction from register a.		
sma	640400	Skip on minus AC	PC + 2→PC if AC_0=1 PC + 1→PC if AC_0=0
	Skip next instruction if AC is minus.		
spa	640200	Skip on plus AC	PC + 2→PC if AC_0=0 PC + 1→PC if AC_0=1
	Skip next instruction if AC is plus.		
spi	642000	Skip on plus IO	PC + 2→PC if IO_0=0 PC + 1→PC if IO_0=1
	Skip next instruction if IO is plus.		
sza	640100	Skip on zero AC	PC + 2→PC if AC=+0 PC + 1→PC if AC≠+0
	Skip next instruction if AC is zero.		

A simple program which illustrates the use of these instructions forms the magnitude of the contents of the PDP-1 accumulator. Figure 6 shows how this operation may be indicated in an algorithm diagram.

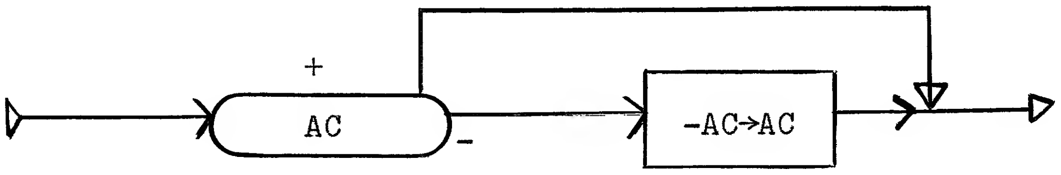


Figure 6--Computation of Magnitude

Two ways of writing the corresponding program are given below.

```
      (a)                             (b)
            20/    sma                      20/    spa
                   jmp  23                         cma
                   cma
```

The second way is preferred as it requires fewer instructions, and less time for execution. In octal notation, these programs would be:

```
      (a)                             (b)
            20/    640400                   20/    640200
                   600023                          761000
                   761000
```

Bit five of a skip instruction will invert the sense of the skip condition when set to one. For example, if we let the symbolic name <u>i</u> have an associated value of 10000_8 , the instruction written as

sza i

will skip when the accumulator does not contain zero. The same rule applies to all skip instructions listed in the above chart.

The Macro Language--The Idiot Multiply Routine

Our next example is an illustration of the MACRO programming language used with the PDP-1 computer. MACRO is a program which will translate an algorithm into the PDP-1 machine language from the symbolic language used for the program examples in these notes. The input for the MACRO conversion program is a paper tape punched with a sequence of 6-bit codes corresponding to the keys of a special typewriter known as a Flexowriter. Each time any key of the Flexowriter is struck, whether alphabetic, numeric, punctuation, or a machine function such as carriage return or tabulate, a distinct 6-bit code is punched in the paper tape. Indeed, the tape may be subsequently read by the Flexowriter, producing the exact same typescript that was produced when the tape was prepared. The output of the MACRO conversion program is a paper tape which may be read directly into the machine without further processing.

To illustrate the format of a source program written in MACRO language we will use an algorithm with two inputs--a quantity x which may be positive or negative, and an integer n (x and n will be the symbols for the registers which contain these quantities). The output will be a quantity s equal to the product of x and n obtained by adding x to itself n times. An algorithm diagram for this algorithm appears in Figure 7.

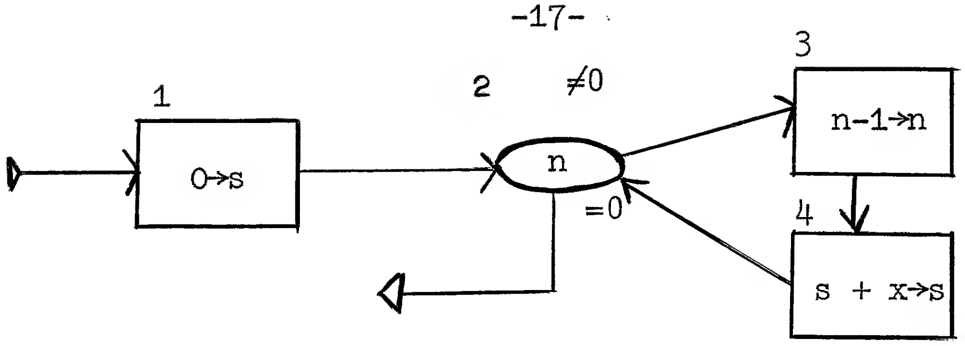


Figure 7--Algorithm Diagram for Idiot Multiply

The PDP-1 program for this algorithm is given below as it would be prepared for conversion by MACRO.

```
Idiot Multiply

40/

beg,      dzm s              - box 1

ret,      lac n       }
          sza i       }      box 2
          jmp end     }

          sub one     }      box 3
          dac n       }

          lac s       }
          add x       }      box 4
          dac s       }

          jmp ret

end,      hlt

x,        0

n,        0

s,        0

one,      +1

start beg
```

The first item on the symbolic program tape must be a title which is followed by a carriage return. Then the body of the program is typed. The last line of the symbolic program is always <u>start</u> <u>z</u> where <u>z</u> designates the address of the first instruction to be executed.

<u>Logical Operations</u>

The PDP-1 Computer includes a group of instructions which perform the common logical operations on each pair of bits from two operand words. The operations are inclusive and exclusive "or" and logical product indicated by the signs ∨, ⊕, and ∧, respectively.

Symbol	Octal Value	Name	Description
ior a	040000 + a	Inclusive or	AC∨CM<a>→AC
	Inclusive or (Unite) contents of register <u>a</u> with accumulator.		
xor a	060000 + a	Exclusive or	AC⊕CM<a>→AC
	Exclusive or (partial add) contents of register <u>a</u> to accumulator.		
and a	020000 + a	And	AC∧CM<a>→AC
	And (logical product) contents of register <u>a</u> with accumulator.		

The complement instruction <u>cma</u>, which was mentioned previously, performs the logical operation of replacing ones by zeros and zeros by ones in the accumulator.

Also included in a group of instructions which rotate the binary digits of the IO, AC, or both combined as shown in Figure 8. The instructions are given in the chart below. In each instruction the number of positions that each bit moves is given by the number of ones in the right hand nine bits of the instruction word. To simplify writing programs, it is convneient to use symbolic names to represent bit congigurations for each possible number of positions of rotation as follows:

Symbol	Octal Value	Positions of Rotation
1s	1	1
2s	3	2
3s	7	3
4s	17	4
5s	37	5
6s	77	6
7s	177	7
8s	377	8
9s	777	9

For describing the rotate operation in the chart, the notation

rr (x,n)

means the result of rotating the quantity x to the right for n positions, etc.

Symbol	Octal Value	Name	Description
rar n	671000 + n	Rotate AC right	rr(AC,n)→AC
ral n	661000 + n	Rotate AC left	rl(AC,n)→AC
rir n	672000 + n	Rotate IO right	rr(IO,n)→IO
ril n	662000 + n	Rotate IO left	rl(IO,n)→IO
rcr n	673000 + n	Rotate combined right	rr(AC,IO,n)→AC,IO
rcl n	663000 + n	Rotate combined left	rl(AC,IO,n)→AC,IO

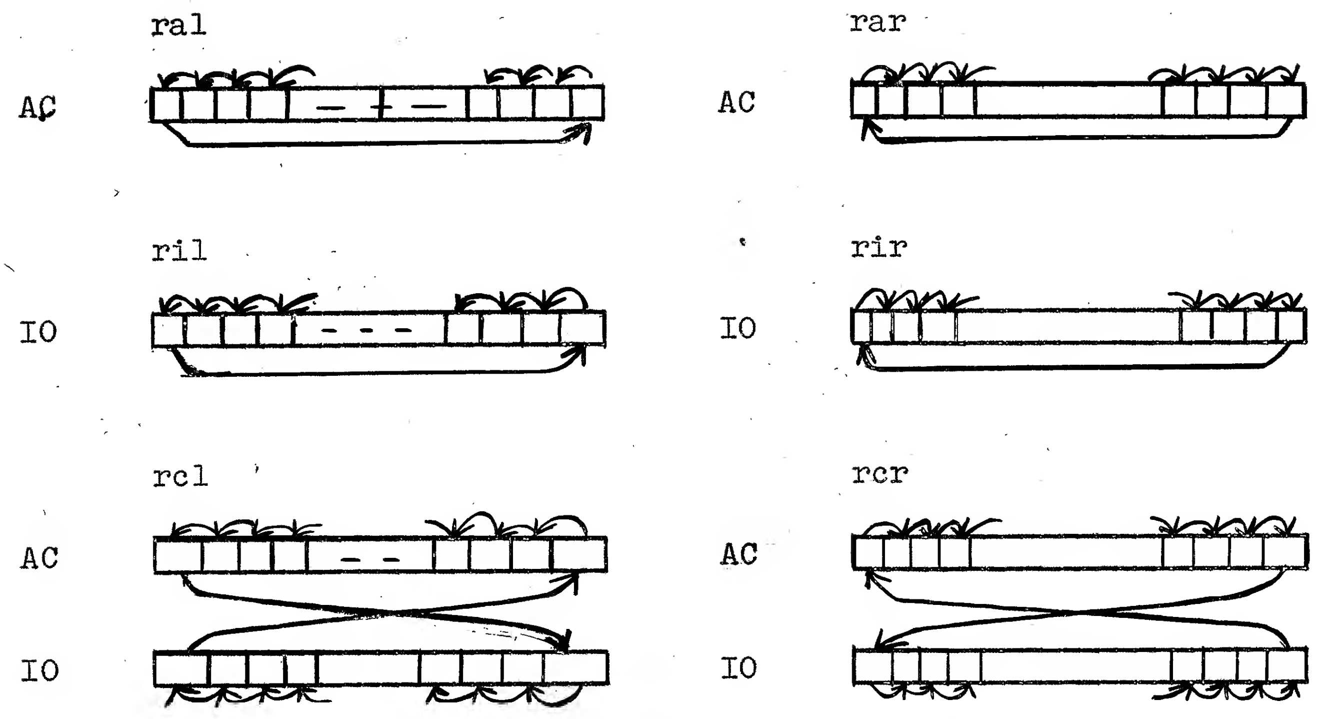


Figure 8--Rotate Configurations

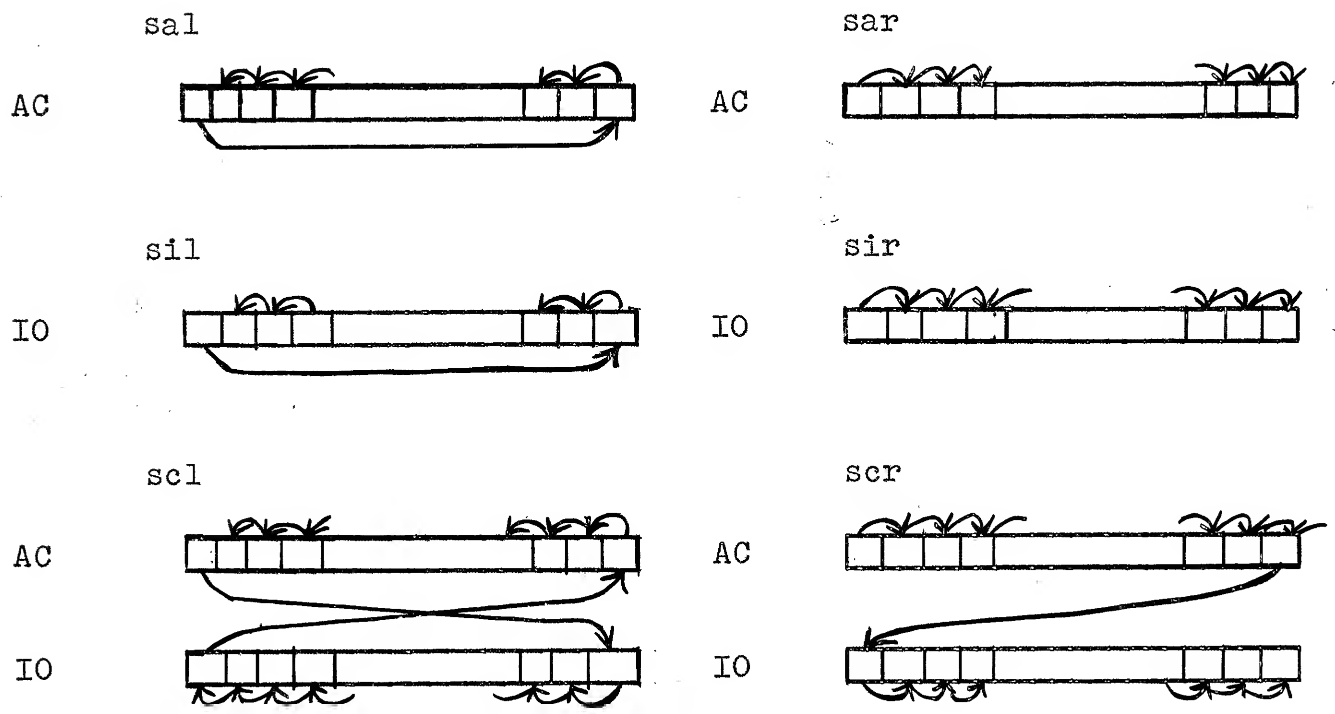


Figure 10--Shift Configurations

A simple illustration of these instructions is a procedure which counts the number of ones in a given word. Let the given word be in register w and suppose the count is to be placed in register n. The algorithm will proceed by examining each bit in succession and setting each one to zero as 1 is found and counted. The algorithm is finished when the word contains all zeros. An algorithm diagram is given in Figure 9.

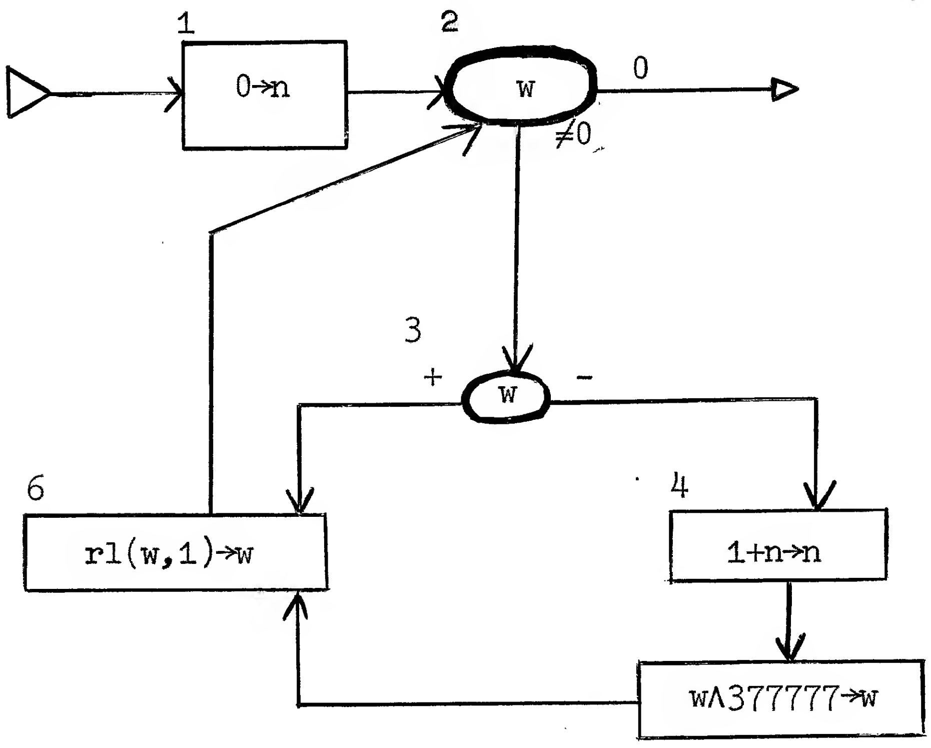


Figure 9--Procedure for Counting Ones in a Word

A PDP-1 program for this procedure is given below:

COUNT BITS

```
     20/
     a,      dzm n       -box 1
             lac w
     b,      sza i       -box 2
             hlt
             sma          box 3
             jmp c
             dac w        box 4
             idx n
             lac w
             and m       -box 5
     c,      ral 1s       box 6
             jmp b
     m,      377777
     w,      0
     n,      0
     start a
```

Shifting

Shifting instructions are built into the PDP-1 computer to permit scaling of quantities, and are arithmetic operations. They are exactly the same as the rotate operations described above with the sole exception that the sign bit remains unchanged. Figure 10 illustrates their operation. On a left shift, the bit immediately to the right of the sign is lost and replaced by its right hand neighbor. The least significant bit is moved to the left and a copy of the sign bit put in its place. The left shift has the effect of multiplying the number represented by two for each position shifted. For negative numbers, this property follows from the observation that complementing may be done either before or after the shift without any effect on the result.

On the right shift command, the most significant bit is replaced by a copy of the sign bit, and the least significant bit is lost. Arithmetically, shifting right amounts to successively dividing by two and rounding down in magnitude.

In a combined shift of AC and IO, the contents of the two registers are treated as a 36-bit one's complement number with sign in bit zero of AC.

The shifting instructions are given in the following chart. As in the rotate instructions, the number of shifts is governed by the number of ones in the right nine bits of the instruction word.

Symbol	Octal Value	Name	Description
sar n	675000 + n	Shift AC right	sr(AC,k)→AC
sal n	665000 + n	Shift AC left	sl(AC,k)→AC
sir n	676000 + n	Shift IO right	sr(IO,k)→IO
sil n	666000 + n	Shift IO left	sl(IO,k)→IO
scr n	677000 + n	Shift combined right	sr(AC,IO,k)→AC,IO
scl n	667000 + n	Shift combined left	sl(AC,IO,k)→AC,IO

Indexing a Table

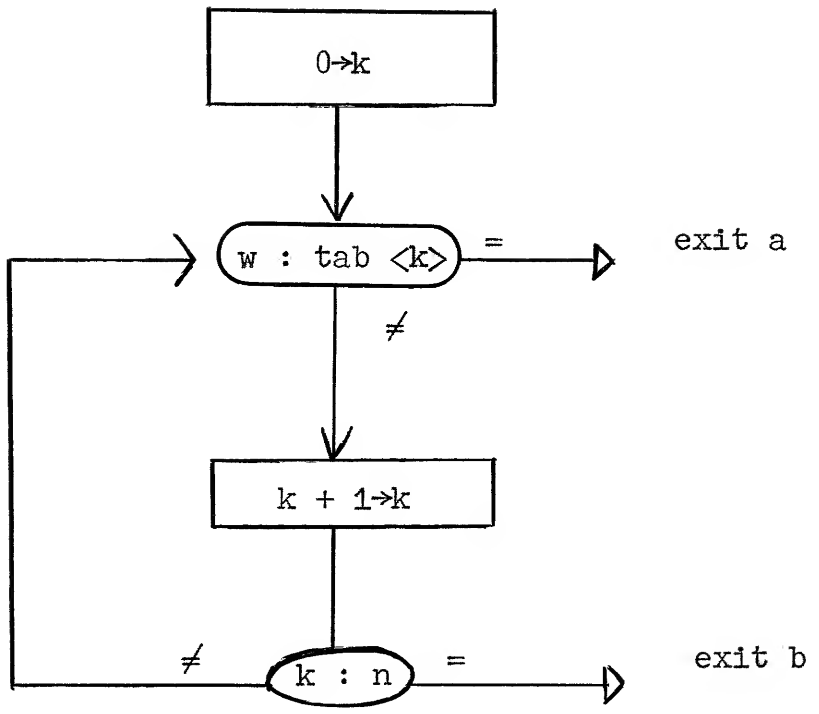
Finding a number of a list of quantities equal to a given quantity is a frequently occurring task in computation. An algorithm for performing this task by a linear search is shown in Figure 11a. Each number of the list tab <k> is compared with the given quantity w until a match if found (exit a) or the list is exhausted (exit b). In the PDP-1 memory we suppose the list is stored with the zeroth entry in the register named tab, and the other entries placed sequentially in memory locations following tab.

The new problem in mechanizing this procedure for the PDP-1 occurs in the comparison step which refers to the k^{th} entry of the list, where k is a variable quantity of the procedure. To facilitate our discussion we may consider the comparison to be done in two steps as shown in Figure 11b: First, the k^{th} table entry is placed in the PDP-1 accumulator; then the decision is made by comparing the contents of register w with the number in the accumulator.

Now, step 2 of the procedure is to place the entry tab <k> in the AC, that is, the contents of the memory register beyond register tab by the value assigned to the quantity k is to be placed in the accumulator. To perform this operation, we would use an instruction whose operation code is lac and address is greater than tab by the value assigned to k. Since the quantity k will take on a succession of values during the course of the procedure, this instruction must also be changed as the computation proceeds. This is possible because instructions are stored in the memory of the computer as configurations of ones and zeros and may be operated on by the computer just as if they represented numerical quantities. To discuss this point in more detail, assume the zeroth table entry is in register 400 of the memory, so that we may associate a value of 400 with the symbolic address tab. At the beginning of the algorithm, k is assigned the value zero, and step 2 is accomplished by the instruction:

lac tab represented by the octal number 200400

(a)



(b)

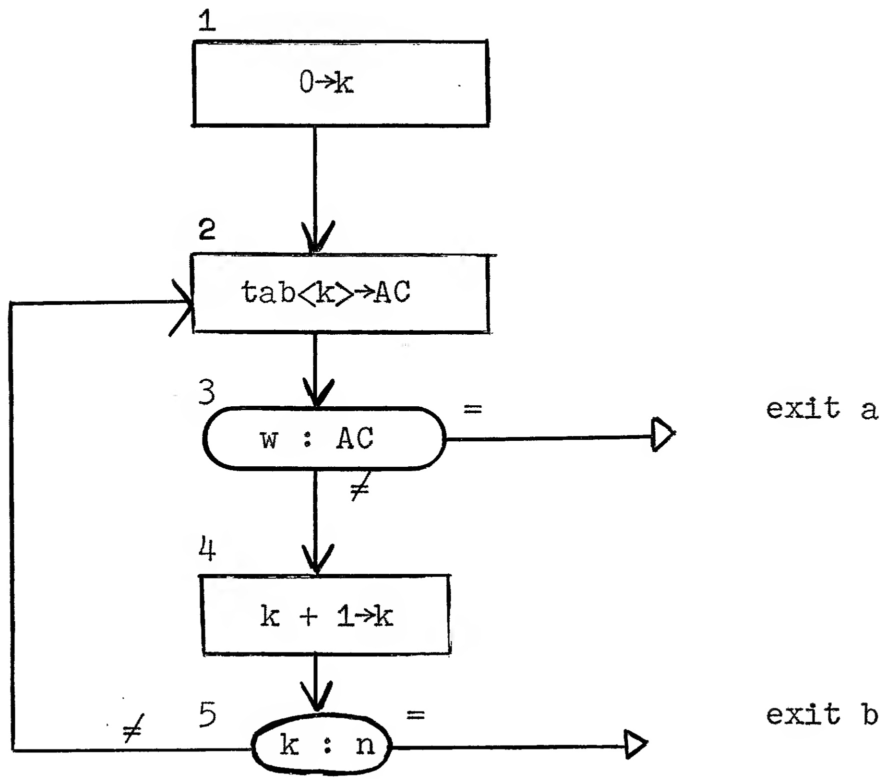


Figure 11--Linear Search Procedure

After step 4 has been performed once, the value assigned to k is one and step 2 is accomplished by the instruction:

```
lac tab+1 or 200401
```

To effect the computation specified in step 2 of the algorithm diagram, we must first <u>calculate</u> and then <u>execute</u> the appropriate instruction. This may be done by the following program steps:

```
s2,     lac k         bse, lac tab
        add bse
        dac ins
ins,  0
```

The line

```
bse, lac+tab
```

indicates that a register whose address is <u>bse</u> contains the instruction <u>lac</u> <u>tab</u> (or the number 200400). To illustrate the operation of this instruction sequence, suppose the value 24 is assigned to the index quantity k. Then, the first instruction will place the number 24 in the accumulator, the second will add the number 200400 leaving the result 200424 in the accumulator, and the third instruction will place this result in register <u>ins</u>. The control element of the computer will immediately read the content of register <u>ins</u> and interpret this as the instruction whose numerical form is 200424, that is the instruction <u>lac</u> <u>tab+24</u>, which places the contents of register <u>tab+24</u> (or register 424) in the accumulator. It is important to realize that the sequence of instructions we write down to form a program only specifies the contents of the computer memory at the beginning of program execution. Specifically, the line

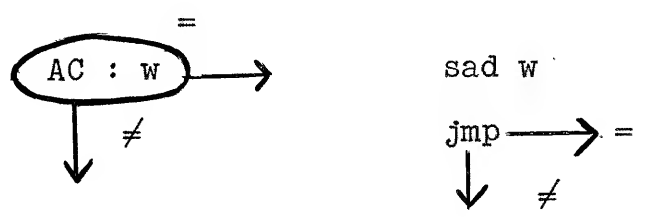
```
ins, 0
```

in the sequence given above means that memory register <u>ins</u> initially contains the number zero. However, the computer does not attempt to interpret the number in register ins as an instruction until it has been changed to 200424 or lac tab+24 through the execution of the preceding instructions.

The comparisons in the linear search algorithm may be performed by the following PDP-1 instructions:

Symbol	Octal Value	Name	Description
sas a	520000 + a	Skip if AC same as	PC + 2→PC if AC=CM<a>
	Skip next instruction if AC is same as contents of memory register a.		
sad a	500000 + a	Skip if AC different	PC + 2→PC if AC≠CM<a> PC + 1→PC if AC=CM<a>
	Skip next instruction if AC is different from contents of memory register a.		

For example, step 3 is accomplished by the following:



The PDP-1 machine has two special instructions to simplify indexing operations as required in step 4:

Symbol	Octal Value	Name	Description
idx a	440000 + a	Index	$CM\langle a\rangle + 1 \rightarrow CM\langle a\rangle, AC$
	Index the contents of register a by one and leave the result also in the AC. A zero result is represented by plus zero.		
isp a	460000 + a	Index and skip on plus AC	$CM\langle a\rangle + 1 \rightarrow CM\langle a\rangle, AC$ $PC + 2 \rightarrow PC$ if $AC_0=0$ $PC + 1 \rightarrow PC$ if $AC_0=1$
	Index the contents of register a by one and skip the next instruction if the result is positive.		

A complete program for the linear search algorithm is given below.

```
Linear search
20/
beg,     dzm k
         lac k
ret,     add bse
         dac ins
ins,     0
         sad w
         hlt           /entry found
         idx k
         sas tst
         jmp ret
         hlt           /entry not in table
bse,     lac tab
tst,     40
k,       0
w,       0
tab,
start beg
```

The alert reader will note that the value assigned to the quantity _k_ is represented in two ways by the above program: It is stored in the usual manner as the contents of a memory register with symbolic address _k_, but it is also represented by the amount that the address portion of the instruction in register _ins_ exceeds the address of the zeroth table entry. It is possible to write a program to mechanize the linear search algorithm in which the value assigned to the quantity _k_ is represented solely by the instruction in register _ins_. Thus, step 1 of the algorithm must assign zero to _k_ by placing the instruction

```
                    lac tab
```

in register _ins_, as follows:

```
               beg, lac set     set, lac tab
                    dac ins
```

The value assigned to k may be indexed by merely indexing the contents of register _ins_. Thus, step 4 becomes:

```
                    idx ins
```

After the last table entry is compared with quantity w, k will be indexed for the 40th time and register _ins_ will contain the instruction

```
               lac tab+40 or 200440
```

in numerical form. This condition may be identified by

```
               sas tst          tst, lac tab+40
```

to accomplish step 5 of the algorithm. The complete program is given on the following page.

Linear Search

```
    20/

    beg, lac set
         dac ins
    ins, 0
         sad w
         hlt        /exit a
         idx ins
         sas tst
         jmp ins
         hlt        /exit b
    set, lac tab
    tst, lac tab+40
    w,   0
    tab,
    tab+40/
    start beg
```

The above program could be further improved through the use of the following PDP-1 instructions:

Symbol	Value	Name	Description
law a	700000+a	Load AC with a	a→AC
	Place the address part of the instruction in the AC with zeros in bits zero through five.		
dap a	260000+a	Deposit address part	$AC_{6-17} \rightarrow CM\langle a\rangle_{6-17}$
	Place the address part of AC contents in the address part of memory register a. The instruction part of CM<a> is not disturbed.		

Thus, the coding of the linear search procedure could begin as follows:

```
                    beg,   law tab
                           dap ins
                    ins,   lac
```

and register set could be omitted as the table base address is contained as the address part of the law instruction. In this case it is necessary to place the operation code lac as the initial contents of register ins as the new initializing sequence only provides the correct address part.

Indirect Addressing

As a second illustration of indexing, consider the algorithm described by Figure 12, which complements the entries of a table.

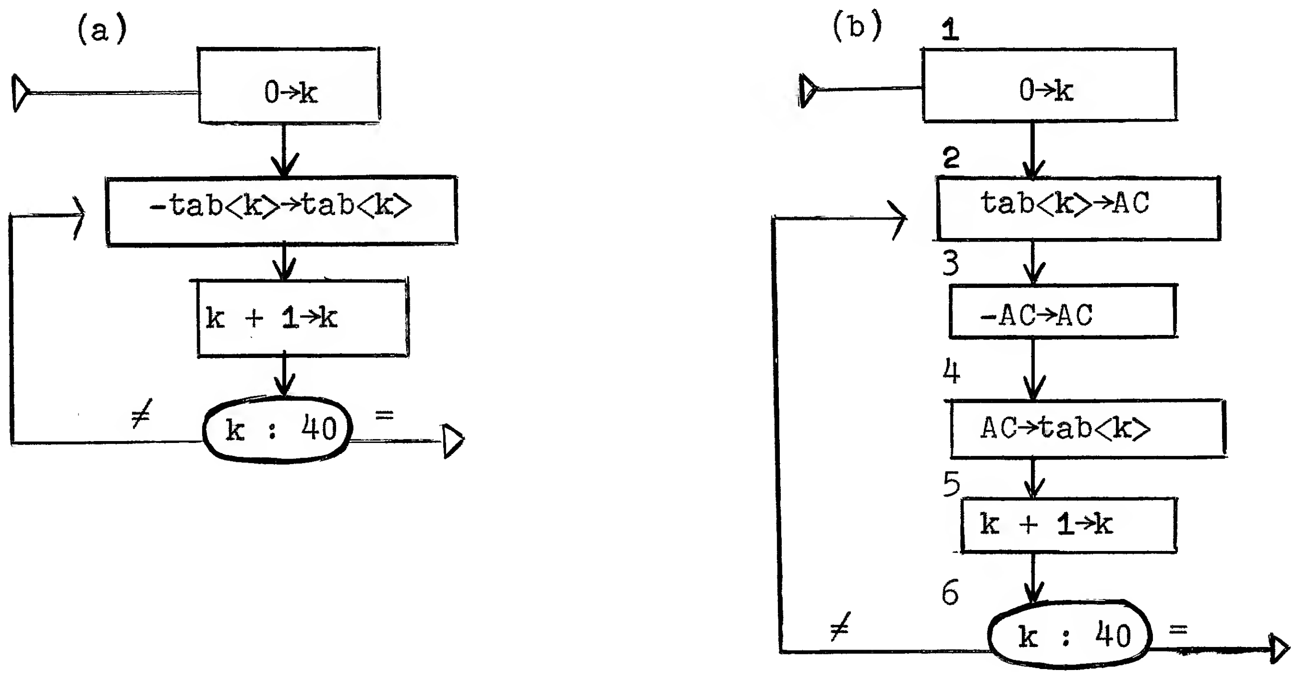


Figure 12--Algorithm to Complement Table Entries

This algorithm makes two references to the k^th^ table entry-- one to read out the entry, and one to store its complemented value. Using the coding principles illustrated in the previous example, we would program this algorithm as follows:

```
Complement Table
20/
beg,      law tab   }
          dap in1   }     box 1
          dap in2   }
in1,      lac           - box 2
          cma           - box 3
in2,      dac           - box 4
          idx in1   }
          idx in2   }     box 5
          sas tst   }
          jmp in1   }     box 6
          hlt
tst,      dac tab+40    - constant
tab,                }
tab+40/             }     space for table
start beg
```

However, the PDP-1 has a feature which permits this process to be coded more compactly. This is <u>indirect addressing</u>. Although this feature applies to nearly all addressable instructions of the PDP-1, we will illustrate it in terms of the <u>dac</u> instruction.

Symbol	Value	Name	Description
dac i a	250000+a	Deposit AC indirect	$AC_1 \rightarrow CM\langle b\rangle$ where $b = CM\langle a\rangle_{6-17}$
	The contents of the accumulator is placed in register b, where <u>b</u> is the address portion of the contents of register <u>a</u>.		

By means of this feature, we need not set up individually all instruction addresses referring to a given table entry. Instead, we may set up one reference, and use indirect addressing for all others. This is illustrated by the recoding of the last example shown below.

```
Complement Table

20/
beg,      law tab }
          dap ins }      box 1
ins,      lac            box 2
          cma            box 3
          dac i ins      box 4
          idx ins        box 5
          sas tst }
          jmp ins }      box 6
          hlt
tst,      lac tab+40     constant
tab,              }
tab+40/           }      space for table

start beg
```

At a point in the algorithm where the index k is assigned a value, say 15, register <u>ins</u> will contain the instruction

```
    lac      tab+15
```

When the instruction

```
    dac i ins
```

is performed, the address part of register <u>ins</u>, that is, tab+15, will be used to select the memory register in which the accumulator content is stored. Thus, the accumulator content becomes the new value of the k^th^ table entry as required by step 4 of the procedure.

Indirect addressing is effected by a one in bit five of the instruction word and may be used with any of the PDP instructions in which the address part selects a memory register.* Of the instructions described so far, this excludes law in which the address part is the operand, and instructions cla, cli, cma, hlt, sza, sma, spa, spi in which the address portion is an extension of the operation code. If we regard the symbol i as having a value of 10000_8, the value of an instruction may still be thought of as the sum of the components of its symbolic representation.

So far we have defined and given an example of single level indirect addressing. If an instruction with an indirect bit addresses a register in which bit five is also one, the indirect addressing is carried to a second level.
Thus if we have

```
ins,   lac i a     a,   i b     b,   c
```

where a, b, and c are symbolic addresses, execution of the instruction in register ins will place the content of register c in the PDP accumulator. This process may be continued for as many levels as desired, although few, if any, programmers have found practical value in more than two levels of indirect addressing.

A second example of the use of indirect addressing is the interchange sort algorithm, which is described in Figure 13. A program for this computation is given on the following page.

*The exceptions to this rule are the instructions jda and cal which are described later in these notes.

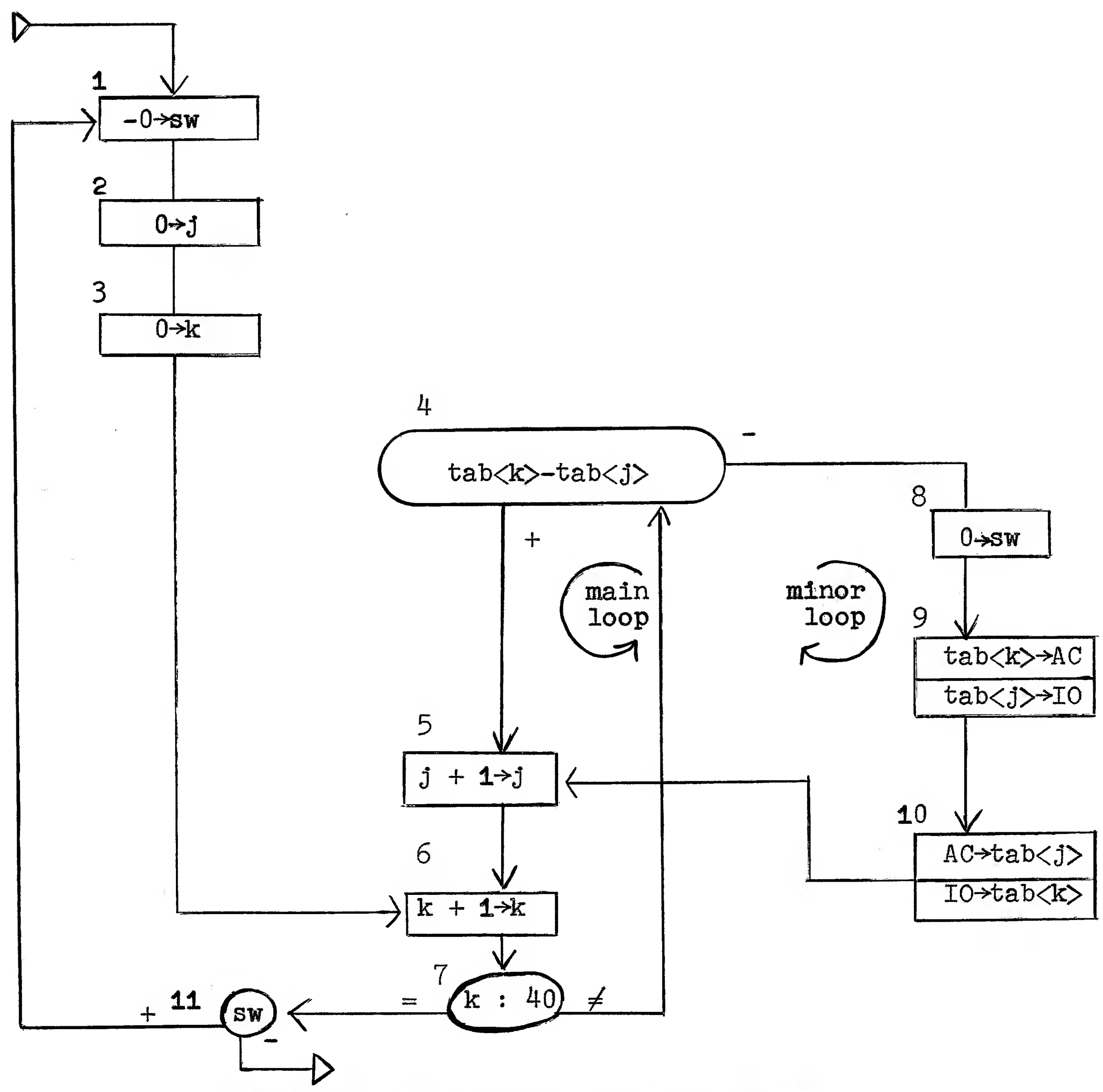


Figure 13--Interchange Sort Algorithm

In this case, the use of indirect addressing simplifies the coding of the minor program loop without placing any extra burden of address setting on the main program loop. This results in a compact program with no sacrifice of speed.

Interchange sort

```
20/
beg,    cla          }
        cma          }    box 1
        dac sw       }
        law tab      }
        dap in1      }    box 2
        dap in2           box 3
        jmp set
in2,    lac          }
in1,    sub          }
        spa          }    box 4
        jmp int      }
ind,    idx in1           box 5
set,    idx in2           box 6
        sas tst      }
        jmp in2      }    box 7
        lac sw       }
        sma          }
        jmp beg      }    box 11
        hlt          }
int,    dzm sw            box 8
        lac i in2    }
        lio i in1    }    box 9
        dac i in1    }
        dio i in2    }    box 10
        jmp ind
tst,    lac tab+40        constant
sw,     0
tab,                 }
tab+40/              }    space for table

start beg
```

<u>Subroutines</u>

Frequently the same computation must be performed at several points in a large program. We may indicate this in an algorithm diagram by giving the computation a name and using a box with the name written inside at points in the diagram where the computation is to be performed. If the computation is very complex, it is desirable to have the group of machine language instructions for its performance stored only once in the computer memory. It is then necessary to transfer control to this group of instructions whenever the mutiply-referenced computation is required by the main program, and to make arrangements to properly return control to the main program when the special computation is finished. A group of instructions which perform a specific operation when called at any point in a main program is called a <u>subroutine</u>.

To illustrate the construction of subroutines, we shall use the very simple computation of forming the sum of two quantities. This computation would not be done by a subroutine in practice as it would be more efficient to write the required instruction sequence each time it is necessary.

Executing the <u>sum</u> computation by a subroutine is indicated in an algorithm diagram as in Figure 14. Here it is assumed that the computation <u>sum</u> is required at two points in the main program.

The double-walled box contains the name of the computation performed by the subroutine and indicates that the following boxes give the steps defining this computation. The box containing the word "return" indicates that the subroutine's computation is completed and control is to return to the next main program step.

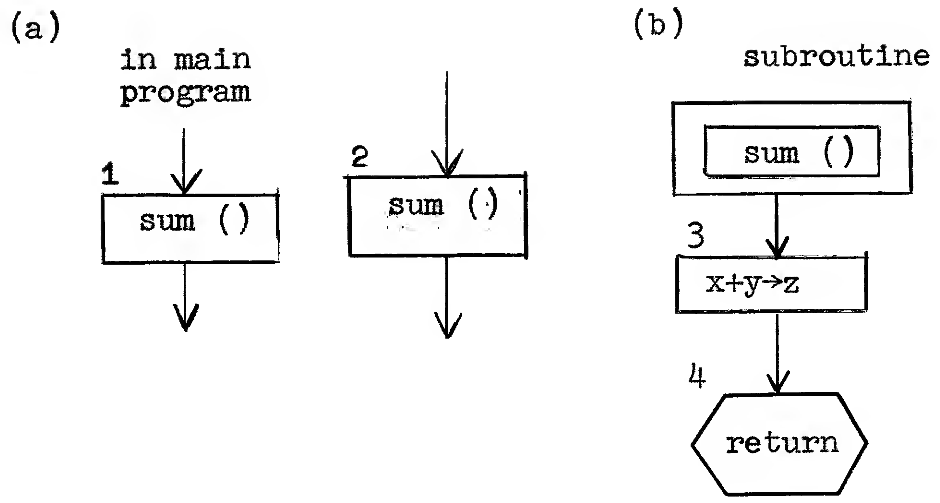
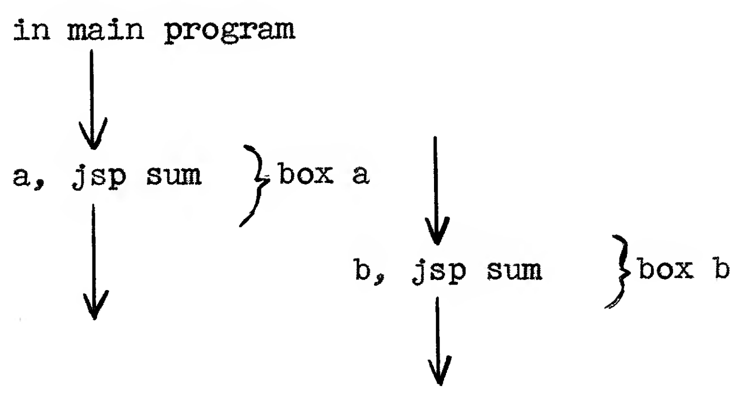


Figure 14--Description of a Simple Subroutine in an Algorithm Diagram

Note that it is necessary to transmit to the subroutine information that will allow the subroutine to return control to the correct point in the main program. All modern computers have special instructions which make this easy to accomplish. In the PDP-1 machine the jsp instruction is used.

Symbol	Value	Name	Description
jsp a	620000+a	Jump save program counter	PC + 1→AC a→PC
	Place the address of the memory register following the one containing the jsp in the AC, and take the next instruction from register a.		

Through the use of jsp instruction, the point of the return to the main program can be given to a subroutine via the PDP accumulator. The program corresponding to the diagram in Figure 14 is given below.



```
subroutine
sum, dap xit      box 1
     lac x  ⎫
     add y  ⎬     box 2
     dac z  ⎭
xit, jmp          box 3
```

When the jsp instruction corresponding to box a of the main program is reached, control is transferred to the instruction in register sum with the address a+1 in the accumulator.

The dap instruction places the address portion of the accumulator content in the address portion of register xit, making the new content of register xit the instruction

jmp a+1

The next group of three instructions performs the computation indicated in box 2. Finally, the instruction in register xit is executed, returning control to the main program at the instruction stored in register a+1 immediately following the jsp instruction which "called" the subroutine. If the subroutine were called by box b, the contents of register xit would become

jmp b+1

and the subroutine would return control to the instruction in register b+1.

Many times, the same computation is desired at several points in an algorithm, but the computation is to be performed on different quantities. Thus, information must be conveyed to the subroutine concerning the quantities involved in the computation. Figure 15 shows how this may be indicated in an algorithm diagram. Here, the symbols x, y and z are dummy names used to define the computation performed by the subroutine. A reference to the subroutine

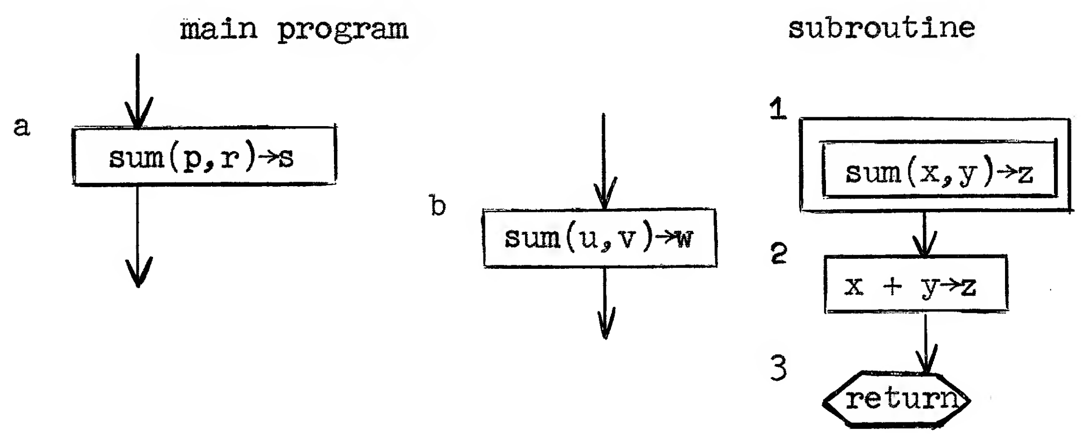


Figure 15--Subroutine Construction by Transmitting Values

from the main program is interpreted by substituting the names of main program quantities for the corresponding dummy names in the definition of the subroutine computation. Information about quantities to be manipulated may be transmitted to the subroutine through the arithmetic registers of the computer. In our example, for instance, the values currently assigned to the quantities substituted for the dummy arguments x and y will be transferred to the subroutine in the AC and IO registers of the machine, respectively. The value computed by the subroutine is to be assigned to the quantity whose name is substituted for the dummy name z. This will be returned to the main program in the PDP accumulator.

In constructing the subroutine to operate in this manner, it is convenient to use the jda instruction of the PDP-1 computer.

Symbol	Value	Name	Description
jda a	170000+a	Jump deposit AC	AC→CM<a> PC + 1→AC a + 1→PC
	Store the AC in register a, place the return address in the accumulator as in the jsp instruction, and take the next instruction from register a+1.		

Using the jda instruction to transfer control to the subroutine, the coding for Figure 15 is as follows:

```
    ↓ main program                                   subroutine
a,  lac p   )                                       sum, 0        )
    lio r   }  box a     ↓                               dap xit  }   box 1
    jda sum )                                            dio y    )
    dac s   )           b,  lac u    )                   lac sum  }
    ↓                       lio v    }  box b            add y    }   box 2
                            jda sum  )              xit, jmp          box 3
                            dac w    )              y,   0
                             ↓
```

Supposing that box a is being executed, control will be transferred to the subroutine at register sum+1 with the value assigned to r in the IO register and that assigned to p in the register sum. The instruction in register xit becomes

jmp a+3

and control returns to the main program with the new value of z in the accumulator.

Another way of constructing the subroutine is to transmit to the subroutine the memory addresses of the quantities involved in the computation. For our example, there would be three data from the subroutine aside from the point of return; these are the memory addresses of the quantities being substituted for the dummy names x, y and z. No data is returned to the main program. Thus, it is appropriate to describe the process as in Figure 16.

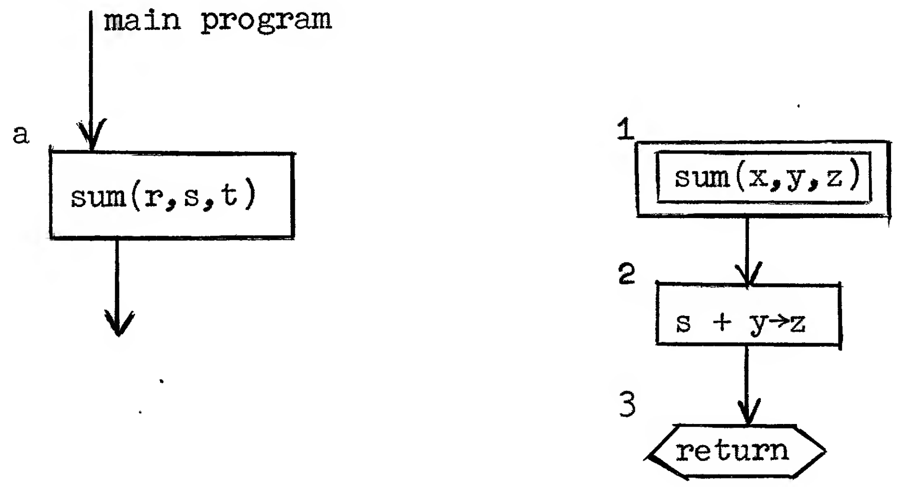
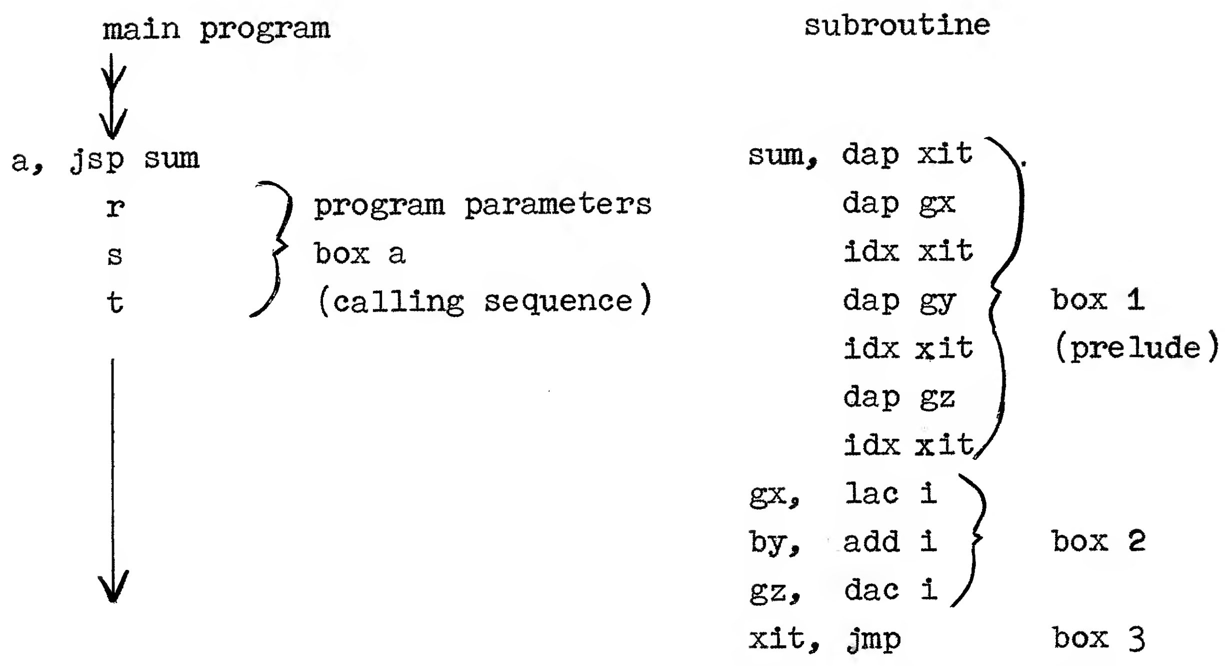


Figure 16--Subroutine Construction by Transmitting Locations

A convneient way of coding this construction makes use of program parameters and is shown on the following page.



```
     main program                              subroutine

a, jsp sum                             sum, dap xit  ⎫
     r          ⎫ program parameters        dap gx   ⎪
     s          ⎬ box a                     idx xit  ⎪
     t          ⎭ (calling sequence)        dap gy   ⎬  box 1
                                            idx xit  ⎪  (prelude)
                                            dap gz   ⎪
                                            idx xit  ⎭
                                       gx,  lac i    ⎫
                                       by,  add i    ⎬  box 2
                                       gz,  dac i    ⎭
                                       xit, jmp         box 3
```

Note that registers a+1, a+2, and a+3 contain the values of the symbols r, s, and t, that is, the memory addresses at which these quantities are stored. If the subroutine is entered from box a, the execution of the instruction marked "prelude" will modify the subsequent instructions of the subroutine to read:

```
gx,    lac i a+1
gy,    add i a+2
gz,    dac i a+3
xit,   jmp a+4
```

Thus, the required computation will be performed by means of indirect addressing, and control will then be returned to the main program at register a+4. The numbers in registers a+1 through a+3 are called <u>program</u> parameters because they are data for the subroutine which may be different at each point from which the main program calls the subroutine. The group of words in the main program required to call a subroutine and apply the necessary information is known as the <u>calling sequence</u> for the subroutine.